Experiment with auto and autograph on a simple case of sliding window protocol

Gérard Boudol, Robert de Simone, Didier Vergamini

INRIA

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.: (1) 39 63 55 11

Rapports de Recherche

N° 870

EXPERIMENT WITH AUTO AND AUTOGRAPH ON A SIMPLE CASE OF SLIDING WINDOW PROTOCOL

Gérard BOUDOL
Robert de SIMONE
Didier VERGAMINI

JUILLET 1988

RR-0870

Experiment with Auto and AutoGraph on a simple case of sliding window protocol.

Experience en Auto et AutoGraph sur un cas simple de protocole à fenêtre coulissante.

Gérard Boudol
INRIA Sophia Antipolis

Robert de Simone
INRIA Sophia Antipolis

Didier Vergamini
CERICS Sophia Antipolis

June 15, 1988

Abstract

Auto is a system designed at manipulations inside the Meije process calculus. It constructs finite automata from relevant terms, and then allow various observational "cross-sections" on the latter to help recognize their good properties (or the lack of them). Autograph pends as its associated graphic interface, by which the user may size terms as visual networks. It also helps depict the resulting automata; which adds the whole thing immediacy. We shall conduct a sample study examplifying Auto and Autograph nowadays use, and raising several further points in future enhancements.

Résumé

Auto est un système de manipulation dans le calcul de processus Meije. A partir de termes appropriés il construit les automates finis correspondants, puis permet des "coupes" observationnelles afin d'y mettre en lumière des propriétés souhaitées, (ou leur échec constaté). Autograph se situe comme son interface graphique, permettant de saisir les termes comme des réseaux visualisés. Autograph aide aussi à dessiner les automates resultants, ce qui facilite l'exploitation rapide des résultats. Nous voulons ici démontrer l'utilité actuelle d'Auto et Autograph à travers un exemple, qui nous permetra aussi de soulever certains points indicatifs de développements prochains.

1 Introduction.

We shall present in this paper a medium-size experience with the Auto and Autograph tools. They shall be applied to a finite-state model of the Stenning protocol in order to assert its correctness.

Auto is a software tool based in Milner's process calculi theory, developed at INRIA. It has been extensively presented in [Ver87]. It allows one to specify networks of communicating processes as terms of an algebra, namely MEIJE. It can perform constructions, reductions, equivalences of underlying automata in the *bisimulation* spirit and provide partial views along *abstraction criteria*. All these preliminaries are supposed familiar to the reader (see [Bou85]).

Autograph is a multi-window graphic interface for Auto: as such it is directed at edition of labelled graphs of course, but of networks of them also, in accordance with process calculi theory. This makes it somehow specific and explains why we did not strive at preexisting graphical systems. Its main features are:

- graphical input by the user of both automata and networks, with –hopefully aesthetic– shortcuts in the graphical grammar to help readability,
- interpretation of the drawings into the Meije algebra interfaced with Auto,
- user-guided output of automata produced by Auto, and soon by other softwares as well (specially the Esterel RealTime language developed in the same project),
- a Postscript output for including drawings in texts.

The Stenning Protocol is a communication protocol, designed at ensuring proper transportation of messages in case of underlying lines which may duplicate, loose or shuffle them. It uses an assumption on messages' life duration, so it can base itself on an acknowledge system running *modulo* a given number. It uses sliding windows: this way messages eventually arrived ahead of the next one expected can be effectively taken into account.

This example was of particular importance to us as its present Meije modelisation made heavy use of parallelism, mostly in between several –more than 2 !– processes at a time. One then encounters a problem of parenthesizing at the best. Treating this example greatly enlarged our intuition. We present some alternative workbenches at the end of the paper.

2 Presentation of the problem.

The so-called Stenning protocol [Ste76] is a data-transfert protocol based on a *sliding window* philosophy. The main interest of this protocol is that required assumptions on media are scarse: lines may duplicate, loose or even deliver messages disordered, and the function of the algorithm is to straighten their flow. The only basic need, as used in the initial proof of correction, is that messages have a bound life duration: once emitted they are either transmitted or lost before this delay has elapsed. One may then tag the messages, and then the corresponding acknowledges, with *modulo integer* values in a certain range, so that these values do not outgrow in space the contents of the messages themselves. Note that in return this requirement is strictly necessary, for no *modulo* labelling could function while arbitrarily old messages could pop up back to the surface and be received at the other end of the line !

We shall not describe formally the protocol in a given "programming" language, but rather assume some reader's acquaintance to it and recall its main features shortly, while its formalisation will be made precise for given window sizes in the Meije modelisation.

There are sliding windows both at the emittor and the receiver. Those windows are intervals suitably chosen, in relation with the *modulo* quotient. Messages and acknowledges may be emitted and received when inside of the window, and also then reemitted on timeouts injonctions. Having a window as large as possible is of interest since then the receiver may start buffering messages ahead of one that has been delayed or forced to be retransmitted. Then a single acknowledge is sent, tagged with the index of the uppermost message received. It validates at once reception of the whole previous initial segment of messages in the emittor window. The basic hypothesis for correction is that the sum of the window sizes, plus the life duration of messages in the lines (we shall suppose only one message is sent at a time) is less than the integer used as *modulo*. A more thorough presentation of the protocol may be found in the litterature [Ste76].

3 Reduction to a given finite case

We shall now proceed with the description of the modelisation of a version of the Stennning protocol in Meije. We choose window sizes of 2 for both the emitter and the receiver, which looked general enough. The life duration has been set to 2 as well, in a sense which will be explained further while describing the lines. Care has to be taken, since this notion ("life length") in our asynchronous setting does not make much sense, just the same as it was already vague and informal in the original presentation. For now we need just mention that this life duration will be represented as the buffering power of the lines, that is the number of succesive messages a given preceding one may survive to. This is the same quantity which was used in the original proof as well.

All these bounds being fixed leave us with a number of 6 for the *modulo*. The exhaustive proof by automaton construction in Auto will let us check the validity of it, under the form of the protocol good functioning.

We shall now display Meije representations of the protocol components, as Autograph drawings. All figures from now on shall be Postscript reprints from automata or nets as they are entered by the user, just slighty polished by the system itself to enjoy the greater accuracy of the printed format.

4 The design of the emitter with AUTOGRAPH.

4.1 Automata

Since we have to label messages modulo a number greater or equal to 6, we chose to build a modular system composed of 6 identical cells, each specialized in the processing of all messages and acknowledges stamped with a given index. These cells are connected in a ring network. We simulate the mechanism of sliding window with internal communications, alike tokens. There are three of them, figuring which cell is in the window, which should read its message from the outside to be transmitted, and which has been acknowledged as well as possibly several others at once. Figure 1 shows us the behaviour of these cells.

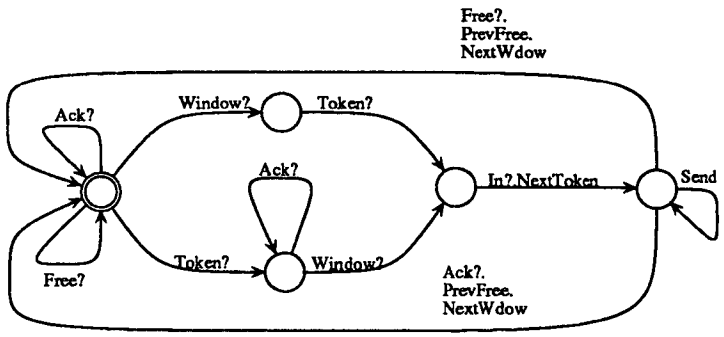

Figure 1: The cell of the emitter

The meaning of each signal is:

- `In` is the signal of input by the environment,
- `Send` is the signal sent to the medium,
- `Ack` is the signal sent by the medium as an acknowledge of the good transition of a message,
- the reception of `Token` allows a cell to take a message from the environment, this permission is then transmitted by the signal `NextToken`,
- the reception of `Window` indicates that the cell is from now in the window where the acknowlegdes are significant,
- the emission of `NextWdow` indicates that the cell leaves the window,

- `PrevFree` is an internal acknowledge transmitted to the preceeding cell in the window, from which it may assume be acknowledged as well, if it needs to,
- `Free` is the reception of this internal acknowledge.

Some labels appear as products (possibly scanning several lines) of the aforementioned signals. Product here means simultaneous (commutative) co-occurring.

In order to initialize the emitter, we must let the two first cells already lay inside the window, same as we must enable the first cell to take its input from the environment. During this initialization step, it is wise to let all previous acknowledges labelled in these cells be ignored, so that the starting point will exactly simulate the meet of an hypothetical preceding warm-up round. So we add the system two starters, dedicated respectively to the proper initiation of cells *one* and cell *two*. number 1. We first focus on the behaviour of starter 1, recalled in figure 2:

- the starter emits asynchronously `Window` and `Token` to the cell,
- the acknowledge from the line are filtered by the starter so they can be transmitted only after it has transmitted `Window` and `Token`.

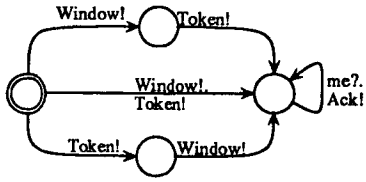

Figure 2: First starter.

The behaviour of the second starter, shown in figure 3, speaks for itself.

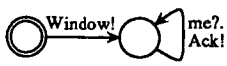

Figure 3: Second starter

4.2 The emitter network.

The six cells and the two starters are lined in a same network (see figure 4): by doing so, we put implicitly their corresponding Meje terms in parallel and we restrict the signals of the global system to only those appearing on the ports of the surrounding box. The communication are figured by the strings connecting ports of several boxes: call "wire" a connected part of ports for the *string attached* relation, then all the signals on a same wire –or "equipotential"– are renamed into the same name. This name is internal (*restricted*) if the wire is not connected to the external box.

4.3 The global system.

The global system for the emitter is obtained from its specification in AUTOGRAPH in three steps:

- the design is translated in a Meije term,
- using AUTO, the term is translated in an automaton, using the `exclusion` function, in which we declare all signals of the emitter to be mutually exclusive (we then get a CCS type of parallel). The purpose of the exclusion function is to build an automaton structurally along the syntax of the term, while propagating to subterms the constraint it was given, thereby diminishing the number of transitions from the very start.

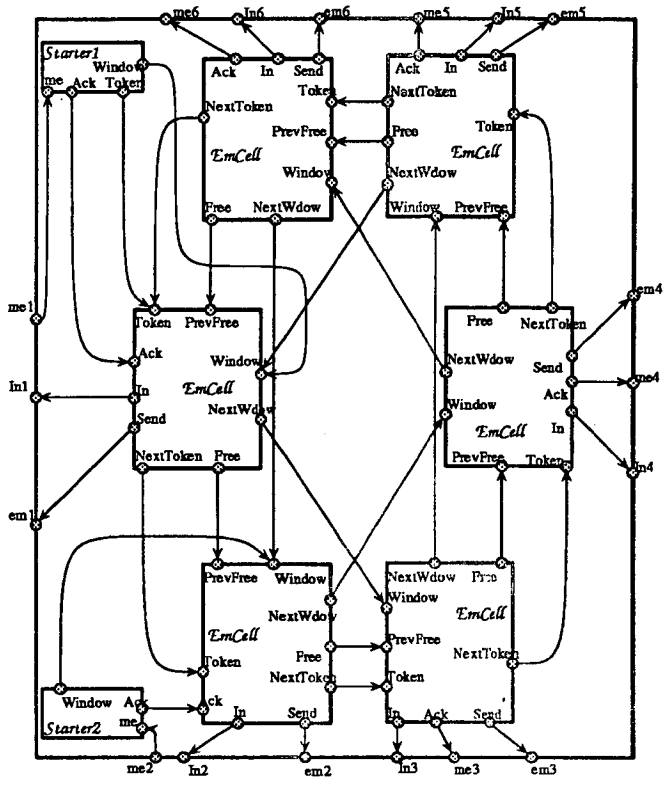

Figure 4: The emitter

- the system is minimized against observational equivalence using the function obs.

Annex A shows the AUTO session of this computation, as well as any other we shall encounter from now on. The size of the final emitter is 18 states and 120 transitions.

5 The design of the receiver with AUTOGRAPH.

5.1 The automata

The receiver is conceived much alike the emitter, with its 6 cells structure. Signals are slightly different:

- `In` is the signal of input by the line,
- `Send` is the signal sent to the environment,
- the reception of `Token` allows a cell to output a message to the environment. This permission is then transmitted to the next cell by the signal `NextToken`,
- the reception of `Window` indicates the cell entered the window, where now acknowledges are significant,
- the emission of `NextWdow` indicates the cell leaving the window.

  Four other internal signals are here to ensure an acknowledge is sent, which is exactly the further acknowledge up the window where receptions of messages were met.
- `NextOk` is a signal sent by a filled cell, ready to output its message.

- `Ok` is the name it is received as in the next cell, which then acknowledges either itself or the preceding one according to whether it has got its own message bufferized or not.
- `PrevOk` is a broadcast message which start scrutation for acknowledgeable cell starting from the one which holds the token allowing output of message...
- `Ack`,`PrevAck`, `NextAck` allow a cell to choose –according to its state and the `Ok` signals received– which acknowledge to send back.

The dubious reader can of course simulate by hand a few steps of the receiver.

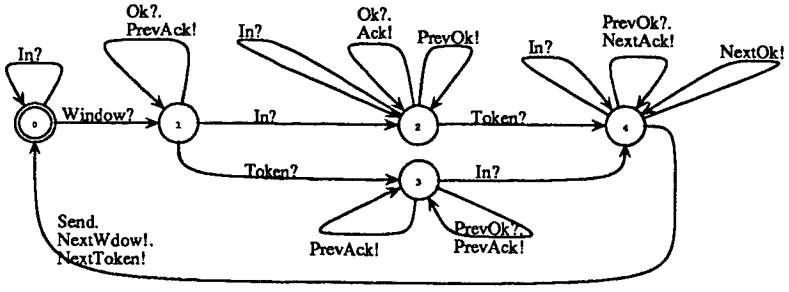

Figure 5: The cell of the receiver

5.2 The receiver network and its underlying global system.

Figure 5 shows us the behaviour of these cells, and figure6 displays the receiver as a net, with starters and wires. It ressembles greatly the emitter. Still one should note the broadcast message `PrevOk`, where the *equipotential* is formed by a reunion of names rather than by pulling strings. We defer explanation of this until the lines description.

Alike the emitter, the AUTO session in A records the receiver construction. It counts 24 states for 180 transitions.

6 The transmission media: Meije models

As mentioned earlier, the Stenning protocol with its *modulo numbers* acknowledges simply could not work without a proper *life length* assumption on messages sent (i.e. they die before a certain time has elapsed). This is obvious, since then any old message with a stamp index equal to a new one inside of window could be substituted from a dormant line cell, which is bad manners. Then no finite representation of the lines could be tempted also.

From the original specification on, it appears the genuine timescale for this delay is the number of messages sent afterwards, which measure the capacity for reordering. From this was borne the idea of modelling this aspect in Meije as a bound on the capacity of media bufferisation mechanisms, so that a given message *has to* be let off the line before the one that is sent the n^{th} afterwards can be allowed in. As communication in between the medium and both the emitter and the receiver are synchroneous, this induces that the message has to be received or purged through the medium before new ones (a certain row later) may be emitted. Significantly, this condition is exactly the one that is needed to ensure a *finite state* modelisation for the whole system.

Thus, both the forward and backward media consist in here of two connected cells, according to the sizes chosen. Messages first embark for the first cell, from which they pass on to the second ones, after which they fumble out of the line itself. During their stay inside the line, they may be variously, and perhaps duplicatively, adressed to the receiver. The various possible connexions result in various hypothesis, each raising to a new possible medium, counting:

- perfect lines,
- lines that may only loose messages (apart from transmitting of course),
- lines that may loose or duplicate messages,

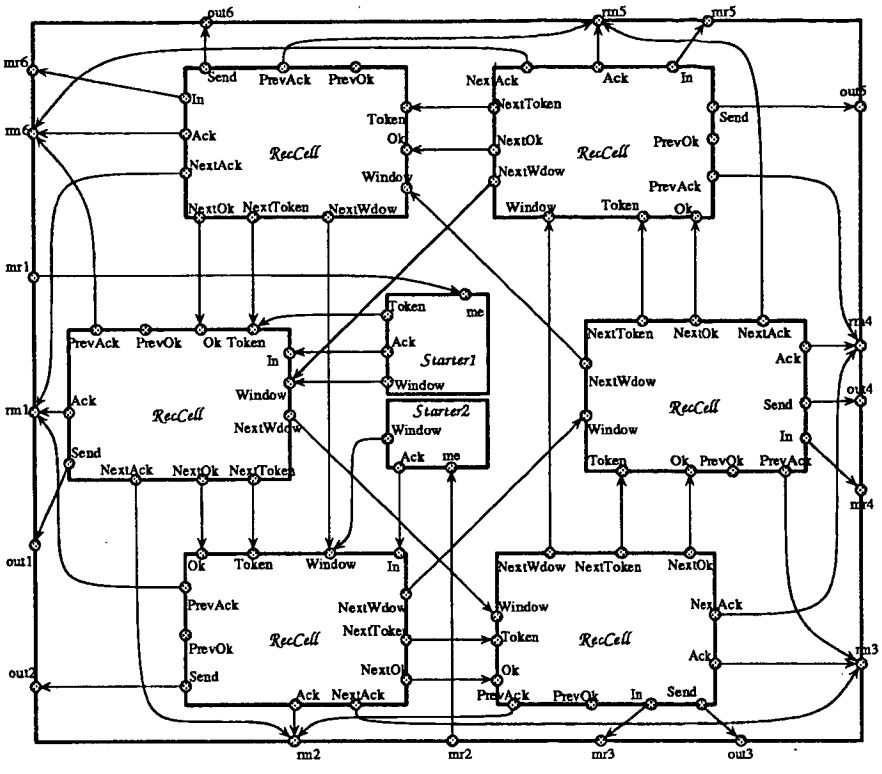

Figure 6: The receiver

- lines that may only duplicate messages,
- lines that may duplicate or permute messages, and finally
- worst lines, that is these that may loose, duplicate and permute messages altogether.

One should bear in mind that permutations or loss of messages should not impair the previous *life length* hypothesis, while they still make sense inside of it.

We shall now present the graphical description of all those lines, in the automatically produced Postscript format. We need comment further on specific drawing conventions that were adopted in Autograph in order to ease the reading of the drawing, that tended to be cramped with too many strings connecting ports. The actual conventions allow a mixed use of strings and explicit renamings which we feel satisfactory. *Satisfaction* here simply means that we do not draw too many strings connecting ports at boxes from different levels of inclusion. The gain is best seen at figure 12.

We call *equipotentials* the various sets of inter-communicating ports in between two boxes levels. An equipotential may of course consists of all the ports that belong to the same connected component for the relation of pulling a string in between ports, but one may also give names to equipotentials, by labelling one of the string of a given connected component, and so all such equally labelled components belong to the same equipotential. Note that a given port labelled α with no string tied to it does belong to the α equipotential, while a port named α with a string pulled to it (which itself is *not* named α !) does *not* ! This last sentence, which may be rephrased as: "a port with no string attached is a connected part on its own", sums up all there is to care about writing equipotentials.

An equipotential may concern one or several ports of the outsider box, in which case a renaming, either alphabetic or not, is performed. If no such ports are involved, then an implicit restriction is assumed on the equipotential.

For the time being the translation into Auto tries and reuses preexisting ports names wherever possible, self-generating a few others for internal use. What will come next is a clever parenthesizing algorithm, that produce

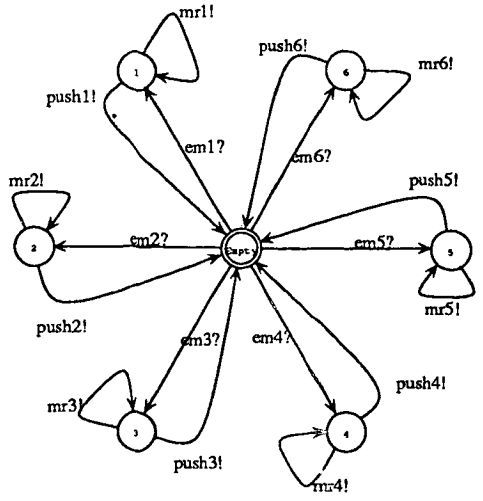

Figure 7: The cell in a line.

binary parallel operators with the least chances of combinatory explosion upon intermediate constructions. It was a major purpose of the case study exposed here to gain insight into which informations were utterly useful for this splitting. All lines use the same line cell (see figure 7).

The actions branded $\mathbf{em}_i$ represent the input of message i. The actions branded $\mathbf{push}_i$ represent the ongoing of message i further along the line. It leaves the cell empty. The actions branded $\mathbf{mr}_i$ represent the output of message i towards the receiver. It is non-destructive and leaves the cell unchanged.

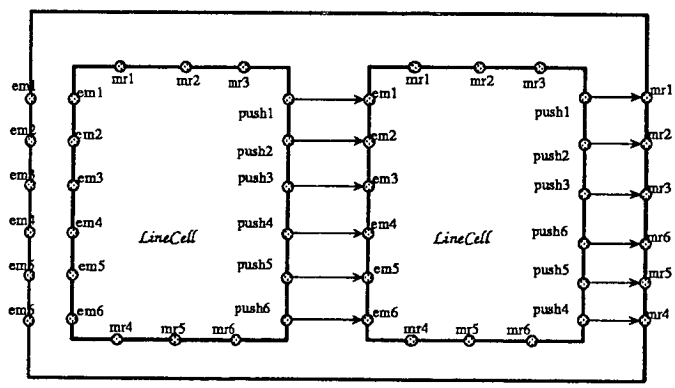

Figure 8: The perfect line.

To build a perfect line (see figure 8), all ports named $\mathbf{mr}_i$ are restricted, as so the sole exit is by being pushed out of the second cell into the receiver. No loss possible indeed.

We build the loosing line (see figure 9) same as first case, but for a sucessive filtering automaton, which may either pass on the message to the receiver, or keep it for itself, that is loose it.

For the duplicating and loosing line (see figure 10), all $\mathbf{push}_i$ i the second cell are wired to an invisible action, so they be lost falling outside range. Connections with the receiver appear only through $\mathbf{mr}_i$ messages from the second cell, thus asserting both duplication and keeping of order. Note the $\mathbf{mr}_i$ mesages from the first cell are implicitly restricted since, although they share names with ports involved each in an equipotential on the second cell, these equipotentials *themselves bear no names !*, (as otherwise they would on the strings).

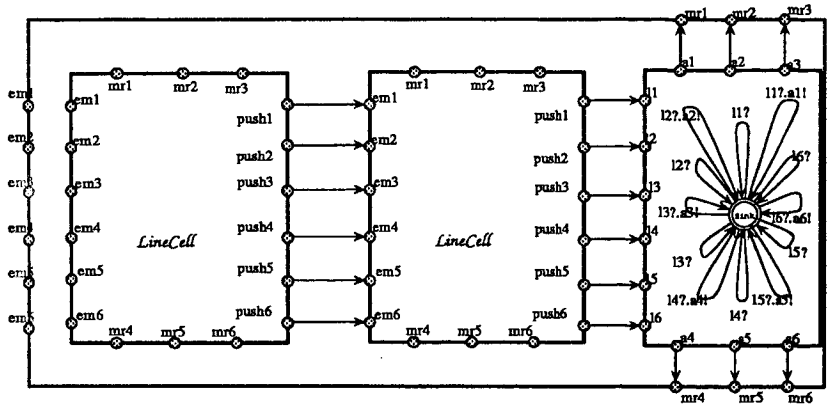

Figure 9: The loosing line.

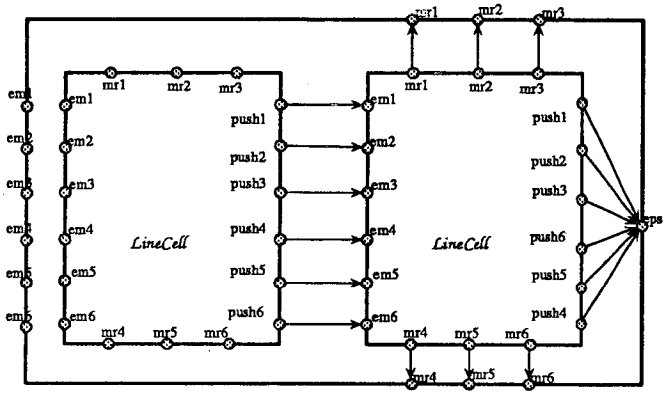

Figure 10: The loosing and duplicating line.

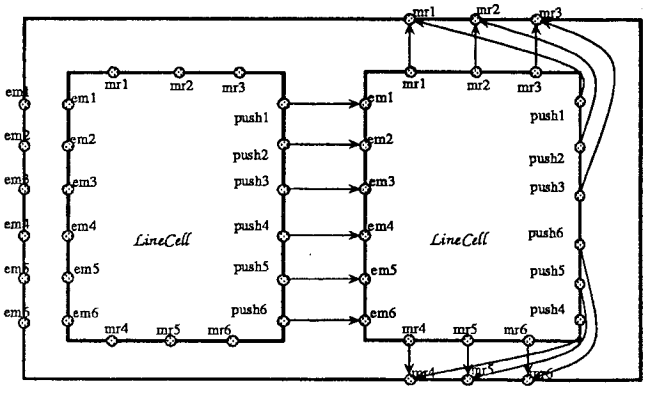

Figure 11: The duplicating line.

The duplicating line is the same as the previous case, except now $\mathtt{push}_i$ messages are turned towards the receiver in order to exit the line, so that no loss remains (see figure 11).

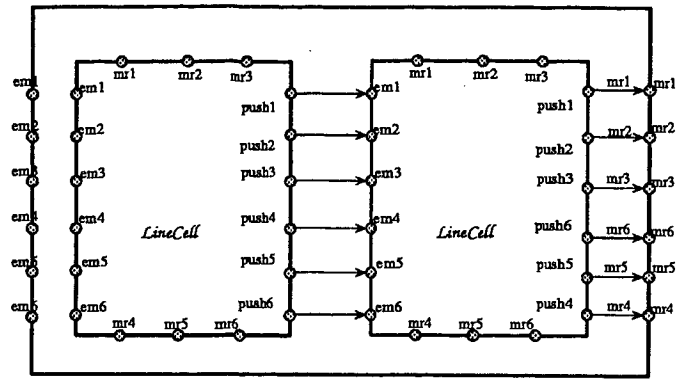

Figure 12: The shuffling and duplicating line.

For the shuffling and duplicating line (see figure 12), all output ports (in a sense informally understandable) are wired to the external $\mathtt{mr}_i$ messages tuned to the receiver.

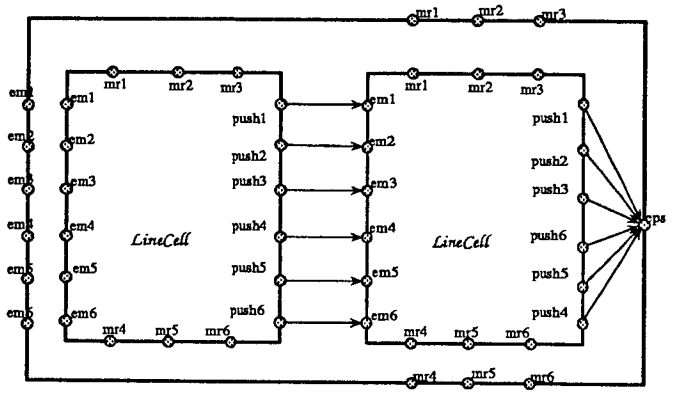

Figure 13: The worst line.

The worst line is the same as the previous one, but for the $\mathtt{push}_i$ messages, which slip off sight (see figure 13).

7 The global net.

It is straightforwardly pictured as its Autograph representation (see figure 14).

The main problem, as far as translation is concerned, is to correctly set parenthesis, linking the emittor either with the forward or the backward line. Here certainly the intuition fails. We tried both approaches and the contrasted results –with all sorts of lines– appear in the sequel. They show a drastic advantage of the combination *emittor/backward line* upon the other solution: in this case both halves get reduced by observational equivalence, while in the other case they do not!

For each type of lines we first give the figures (time of construction and size in *states/transitions*) for the lines themselves, putting together the two cells as prescribed. Then we unravel the numbers for the alternative nets.

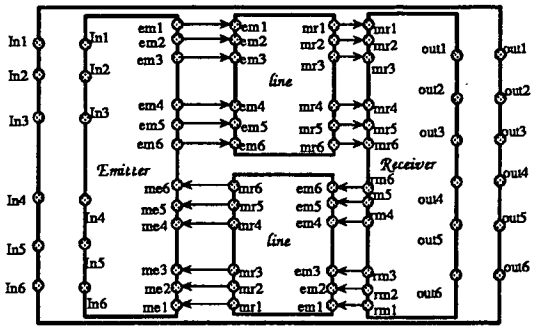

Figure 14: The global net.

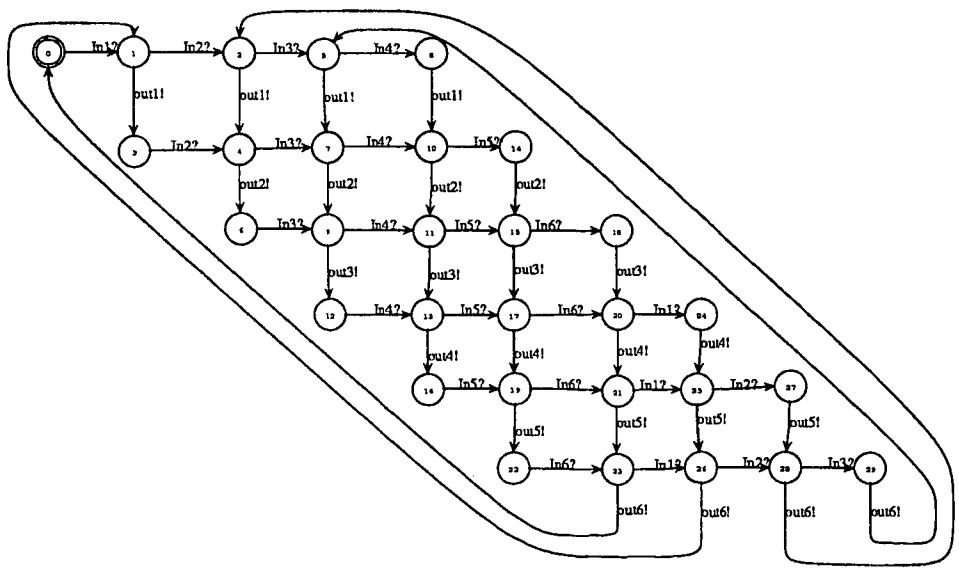

Figure 15: The specification.

All final results comes down onto the same automaton, in fact. This automaton, considered as a goal for specification, can be best obtained by wiring the emittor straight into the receiver. It is a four slots buffer, as cells from the emittor may preempt their messages in advance. We used Autograph to output it graphically in figure 15 (using a feedback from Auto with a menu function called **explore**). The positioning of states is man-made with progressive exploration of next neighbours as the system progressively encounters them. This may seem disappointingly crude, but we certainly did not want to get pointlessly involved into automatic displays of graphs. Here again, treatment of examples like Stenning protocol will expectingly provide hints towards realistic, insightful partial machine assistance.

Figure 16 shows all the size and computation time of the automata obtained by AUTO on a computer Gould PN9080. In each column we indicate the size of the system before and after reduction and the global time of both construction and reduction. We can see that it is better to put together the emmitter with the backward line than with the forward line because each "half" of the net is reduced in the first case giving a much smaller global system.

		EM+BackL		REC+ForwL		global	EM+ForwL		REC+BackL		global
Perfect	43	774	162	1032	162	540	774	774	1032	1032	5364
Line	84	2676	666	3564	762	2184	5268	5268	7884	7884	24972
		41 sec		79 sec		30 sec	56 sec		69 sec		934 sec
Loosing	43	774	180	1032	174	414	774	726	1032	978	1392
Line	142	4836	1314	6108	1350	2316	10416	9642	15588	14622	9378
		58 sec		90 sec		18 sec	89 sec		204 sec		247 sec
Loosing	37	666	180	888	174	414	666	636	888	858	1014
Duplicating	144	4674	1314	6084	1350	2316	8958	8442	13404	12822	7002
Line		52 sec		71 sec		31 sec	75 sec		182 sec		136 sec
Duplicating	43	774	162	1032	162	540	774	774	1032	1032	5436
Line	126	3306	666	4572	762	2184	6024	6024	8892	8892	32022
		53 sec		69 sec		42 sec	46 sec		82 sec		1149 sec
Shuffling	43	774	342	1032	282	552	774	774	1032	1032	8442
Duplicating	168	3540	1356	4800	1326	2268	6780	6780	9900	9900	48000
Line		64 sec		99 sec		43 sec	87 sec		126 sec		*aborted*
Worst	43	774	324	1032	294	414	882	726	1176	972	3366
Line	234	5682	2316	7320	2154	2382	13260	10974	19620	16344	30246
		73 sec		110 sec		21 sec	74 sec		129 sec		*aborted*

Figure 16: All the results obtained by AUTO.

8 Conclusions

We conducted an experiment with Auto and Autograph on the well-known example of the Stenning protocol, and proved that its functioning remained unaltered while changing communication media amongst several. The results were established only for fixed values of the parameters since the techniques rely on finite automata constructions. Although one may feel it is a strong limitation, one should alredy suspect the usefulness –certainly underestimated in the previous example– for analysing systems which need not be correct at all ! One may then reduce them, get partial views on them (by tracing some signals) and graphically explore their states space, until figuring what goes wrong.

One may also compare a realisation with its implementation, as we showed all along. Now the tool should be developed towards a better diagnostic of non-equivalence in case of failure. Here is certainly a motive for animation in Autograph. Separating temporal logic formulae could certainly also be considered.

Other future directions include the extension of the tools to other algebrae, like Lotos for instance. A possibility of user-defined algebra is even foreseen, using Ecrins for semantic definition of operators by conditional rewrite rules. Extensions in the realm of shorthand notations, like linked structures of processes parametered on concrete

indexes -arrays, rings, so on ...-, would also be beneficial. But at they stand, we feel Auto and Autograph to be of great help in formulating and analysing any problems of finite state nature concerned with process algebra.

References

[Bou85] G. Boudol.
Notes on algebraic calculi of processes.
In K. Apt Editor, editor, *Logics and Models for Concurrent Systems*, Springer-Verlag, 1985.

[Ste76] N. V. Stenning.
A data transfert protocol.
Computer Networks, 1:99–110, 1976.

[Ver87] D. Vergamini.
Vérification de réseaux d'automates finis par équivalences observationnelles: le système AUTO.
Thèse de doctorat, Université de Nice, 1987.

A The Auto session.

```
                                AUTO
                        Version 1.2 (Janvier 88)

@ load "EmCellMetro";
@ parse EmCell =
meije0>    let rec
meije0>      {st_0 = Token?:st_1 + Window?:st_2 + Ack?:st_0 +
meije0> Free?:st_0
meije0>    and
meije0>      st_1 = Window?:st_3 + Ack?:st_1
meije0>    and
meije0>      st_2 = Token?:st_3
meije0>    and
meije0>      st_3 = In?.NextToken:st_4
meije0>    and
meije0>      st_4 = Ack?.PrevFree.NextWdow:st_0 + Send:st_4 +
meije0>             Free?.PrevFree.NextWdow:st_0}
meije0>    in st_0;
EmCell : Process of meije0

time = 0.66s
@ file ./EmCellMetro.ec loaded.

time = 0.08s
@ load "Starter1";
@ parse Starter1 =
meije0>    let rec
meije0>      {st_0 = a!:st_1 + b!:st_3 + a!.b!:st_2
meije0>    and
meije0>      st_1 = b!:st_2
meije0>    and
meije0>      st_2 = d?.c!:st_2
meije0>    and
meije0>      st_3 = a!:st_2}
meije0>    in st_0;
Starter1 : Process of meije0

time = 0.46s
@ file ./Starter1.ec loaded.

time = 0.06s
@ load "Starter2";
@ parse Starter2 =
meije0>    let rec
meije0>      {st_0 = a!:st_1
meije0>    and
meije0>      st_1 = d?.c!:st_1}
meije0>    in st_0;
Starter2 : Process of meije0

time = 0.28s
@ file ./Starter2.ec loaded.

time = 0.08s
@ load "EmitMetro";
@ parse Emitter =
meije0>       % local signals sig_1 sig_2 sig_3 sig_4 sig_5
meije0> %        sig_6 sig_7 sig_8 sig_9 sig_10 sig_11 in
```

```
meije0> ((Starter2 [Window /a,me2/d])
meije0> //
meije0> %*********************************************************
meije0>       (((EmCell [sig_1 /NextToken, NextToken /Token, em2
meije0>  /Send, In2 /In, c /Ack])
meije0>       //
meije0>       (EmCell [sig_5 /NextToken, sig_1 /Token, sig_4 /
meije0> NextWdow, sig_3 /Window, Free /PrevFree, sig_2 /Free,
meije0> em3 /Send, In3 /In, me3 /Ack]))\Free\sig_1
meije0>       //
meije0>       (EmCell [sig_8 /NextToken, sig_5 /Token, sig_7 /
meije0> NextWdow, NextWdow /Window, sig_2 /PrevFree, sig_6 /
meije0> Free, em4 /Send, In4 /In, me4 /Ack]))\sig_2\sig_5\NextWdow
meije0> %*********************************************************
meije0> //
meije0> %*********************************************************
meije0>       ((EmCell [sig_3 /NextWdow, a /Window, sig_9 /
meije0> PrevFree, PrevFree /Free, em1 /Send, In1 /In, c /Ack]
meije0>        //(Starter1 [Token /b, me1/d]))
meije0> //
meije0>       (((EmCell [sig_11 /NextToken, sig_8 /Token, a /
meije0> NextWdow, sig_4 /Window, sig_6 /PrevFree, sig_10 /Free
meije0> , em5 /Send, In5 /In, me5 /Ack])
meije0>       //
meije0>       (EmCell [Token /NextToken, sig_11 /Token, Window
meije0>  /NextWdow, sig_7 /Window, sig_10 /PrevFree, sig_9 /
meije0> Free, em6 /Send, In6 /In, me6 /Ack]))\sig_11\sig_10))\Token\sig_9\a\c
meije0> %*********************************************************
meije0> )\sig_8 \sig_7 \sig_6 \sig_4 \sig_3 \NextToken \PrevFree \Window \c;
Emitter : Process of meije0

time = 2.64s
@ file ./EmitMetro.ec loaded.

time = 0.10s
@ set Emit = exclusion (Emitter,
@      {{In1, em1, me1 ,
@        In2, em2, me2 ,
@        In3, em3, me3 ,
@        In4, em4, me4 ,
@        In5, em5, me5 ,
@        In6, em6, me6 }});
Emit : Automaton

time = 46.54s
gc= 1
@
@ set EMIT=obs Emit;
EMIT : Automaton

time = 0.50s
@ display it short;
size = 18 states, 120 transitions, 19 actions.

time = 0.18s

@ load "RecCell";
@ parse RecCell =
meije0>    let rec
meije0>      {st_4 = Send.NextToken!.NextWdow!:st_0 + In?:st_4 + NextOk!:st_4 +
```

```
meije0> PrevOk?.NextAck!:st_4
meije0>    and
meije0>      st_3 = In?:st_4 + PrevOk?.PrevAck!:st_3 + PrevAck!:st_3
meije0>    and
meije0>      st_2 = Token?:st_4 + In?:st_2 + Ok?.Ack!:st_2 +
meije0> PrevOk!:st_2
meije0>    and
meije0>      st_1 = In?:st_2 + Token?:st_3 + Ok?.PrevAck!:st_1
meije0>    and
meije0>      st_0 = Window?:st_1 + In?:st_0}
meije0>    in st_0;
RecCell : Process of meije0

time = 0.86s
@ file ./RecCell.ec loaded.

time = 0.10s
@ load "Starter1";
@ parse Starter1 =
meije0>    let rec
meije0>      {st_0 = a!:st_1 + b!:st_3 + a!.b!:st_2
meije0>    and
meije0>      st_1 = b!:st_2
meije0>    and
meije0>      st_2 = d?.c!:st_2
meije0>    and
meije0>      st_3 = a!:st_2}
meije0>    in st_0;
Starter1 : Process of meije0

time = 0.46s
@ file ./Starter1.ec loaded.

time = 0.06s
@ load "Starter2";
@ parse Starter2 =
meije0>    let rec
meije0>      {st_0 = a!:st_1
meije0>    and
meije0>      st_1 = d?.c!:st_1}
meije0>    in st_0;
Starter2 : Process of meije0

time = 0.28s
@ file ./Starter2.ec loaded.

time = 0.06s
@ load "RecMetro";
@ parse Receiver =
meije0> (
meije0> %*************************************************************
meije0>       (((((RecCell [sig_7 /NextWdow, sig_8 /NextOk, sig_9 /
meije0> NextToken, b /Window, NextOk /Ok, NextToken /Token,
meije0> out1 /Send, rm6 /PrevAck, rm1 /Ack, rm2 /NextAck])
meije0>      //(Starter1 [In /c, mr1 /d, NextToken /a]))\In
meije0>     //
meije0>       ((RecCell [c /In, sig_4 /NextWdow, sig_10 /NextOk
meije0> , sig_11 /NextToken, NextWdow /Window, sig_8 /Ok, sig_9
meije0>  /Token, out2 /Send, rm1 /PrevAck, rm2 /Ack, rm3 /
meije0> NextAck])// (Starter2 [mr2 /d, NextWdow /a]))\c )\sig_8\sig_9)
```

```
meije0>     //
meije0>     (RecCell [mr3 /In, sig_1 /NextWdow, sig_5 /NextOk
meije0> , sig_6 /NextToken, sig_7 /Window, sig_10 /Ok, sig_11 /
meije0> Token, out3 /Send, rm2 /PrevAck, rm3 /Ack, rm4 /NextAck]))\sig_7\sig_11\sig_10

meije0> %************************************************************
meije0> //
meije0> %************************************************************
meije0>       ((((RecCell [mr4 /In, Window /NextWdow, sig_2 /
meije0> NextOk, sig_3 /NextToken, sig_4 /Window, sig_5 /Ok,
meije0> sig_6 /Token, out4 /Send, rm3 /PrevAck, rm4 /Ack, rm5 /
meije0> NextAck])
meije0> //
meije0>       (RecCell [mr5 /In, b /NextWdow, Ok /NextOk, Token
meije0>  /NextToken, sig_1 /Window, sig_2 /Ok, sig_3 /Token,
meije0> out5 /Send, rm4 /PrevAck, rm5 /Ack, rm6 /NextAck]))\sig_3\sig_2)
meije0> //
meije0> (RecCell [mr6 /In, out6 /Send, rm5 /PrevAck, rm6 /Ack
meije0> , rm1 /NextAck]))\Ok\Token\Window
meije0> %************************************************************
meije0> )\sig_6\sig_5\sig_4 \sig_1 \b \PrevOk \NextWdow \NextOk \NextToken
meije0>
meije0> ;
Receiver : Process of meije0

time = 2.72s
@ file ./RecMetro.ec loaded.

time = 0.06s
@ set Recep = exclusion (Receiver,
@      {{out1, rm1, mr1 ,
@        out2, rm2, mr2 ,
@        out3, rm3, mr3 ,
@        out4, rm4, mr4 ,
@        out5, rm5, mr5 ,
@        out6, rm6, mr6 }});
Recep : Automaton

time = 39.12s
gc= 1
@
@ set RECEP=obs Recep;
RECEP : Automaton

time = 0.56s
@ display it short;
size = 24 states, 180 transitions, 19 actions.

time = 0.20s

@ load "specdemo";
@ parse specif =
meije0>              (EMIT[mr1/em1,mr2/em2,mr3/em3,
meije0>                  mr4/em4,mr5/em5,mr6/em6]
meije0>              //
meije0>               RECEP[me1/rm1,me2/rm2,me3/rm3,
meije0>                     me4/rm4,me5/rm5,me6/rm6]
meije0>              )\me1\me2\me3\me4\me5\me6\mr1\mr2\mr3\mr4\mr5\mr6;
specif : Process of meije0
```

```
time = 0.66s
@
@
@ set specif = obs exclusion(specif,{{In1,In2,In3,In4,In5,In6,
@                                    out1,out2,out3,out4,out5,out6}});
specif : Automaton

time = 5.20s
@ display it short;
size = 30 states, 48 transitions, 13 actions.

time = 0.22s
@
@ file ./specdemo.ec loaded.

time = 0.06s
```

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique